REMARKS

This is intended as a full and complete response to the Final Office Action dated October 20, 2006, having a shortened statutory period for response set to expire on January 20, 2007. Applicants submit this response to place the application in condition for allowance or in better form for appeal. Please reconsider the claims pending in the application for reasons discussed below.

Claims 1, 2, 4-6, 8-12 and 14-20 are pending in the application and remain pending following entry of this response. Claims 3, 7 and 13 have been cancelled. Claims 9 and 16 have been amended to recite a "storage" medium. Applicants submit the amendment does not introduce new matter or raise new issues.

Claim Rejections - 35 U.S.C. § 103

Claims 1-20 stand rejected under 35 U.S.C. § 103(a) as being unpatentable over *Wimble* (U.S. Pat. 5,812,850) in view of *Warmink* (U.S. Pat. 6,611,924). Applicants respectfully traverse this rejection.

Wimble discloses "a debugging system which provides an interactive and dynamic environment for computer program debugging." *Wimble*, 1:18-20. "The debugging system uses a database of information relating machine executable code to source code. The database is developed during the compilation process using an extensible object-oriented set of tools." *Wimble*, Abstract. The debugger disclosed in *Wimble*

> generates *information to be used by the debugger* in a Debugging System. The information may be in the form of data bases. Debugging information is really a database of information about a compiled program.

Wimble, 6:30-34. *Wimble* is very specific regarding the information stored in the debugger database. In particular, *Wimble* discloses that information in the debugger database includes "a collection of Symbolic Elements 417 which is to be built, and

maintain *a description of the program once it has been built*." *Wimble*, 8:27-37. As the emphasized passage makes clear, the debugger database of *Wimble* is created, and used, by the debugger to assist in the debugging process, and is not accessed by the program being debugged.

Nevertheless, the Examiner continues to assert that *Wimble* discloses steps of a method of debugging executable code configured to access associated data in a data repository. Applicants respectfully disagree. As claimed, the method includes a step of determining whether the monitored executable code has accessed the associated data in the data repository, That is, the claim recites a step of determining whether the code being debugged access data from a data repository as part of the execution of the executable code. To support the rejection, present rejection asserts that Wimble discloses the step of "determining whether the monitored executable code has accessed the associated data in the data repository (Col12:47-67, ' ... a component in the Debugger Database ...') " *Final Office Action*, p. 3. Respectfully, the Examiner's argument is untenable, as there is no readily apparent relationship between "a component in the debugger database" and the claimed method step of "determining whether the monitored executable code has accessed the associated data in the data repository." Further, the program being debugged using the System of *Wimble* wouldn't even access the debugging information used by the debugger. Set out in full, the cited passage provides:

> ADOPTPCSOURCEMAP
>
> FIG. 11 is a block diagram of the function AdoptPCSourceMap 129 for asking the information reader to adopt a particular format of information from a provider. The provider must put together a list of PCSourceMapElement entries 130 and ask the Debugger to adopt the information which describes a component in the Debugger Database. The following shows the Adopt function interface:
>
> ---
>
> void AdoptPCSourceMap (const THoopsPropertyName name, TPCSourceMapList* pcSourceMap);

> There is a different map kept to describe the Interface v. the Implementation properties of a component, so the developer must supply a "name" parameter to indicate which property the map describes.
>
> THE TOKEN MAP
>
> FIG. 12 is a block diagram showing a general overview of Token Map 66, and details of the Token Map entry objects 78.

Plainly, nothing in this passages discloses the step of determining whether the code being debugged access data from a data repository. Instead, the passage describes the use of function call "AdoptPCSourceMap" to ask an "information reader" to adopt a particular format of information from a provider. For all these reasons, Applicants submit that while *Wimble* discloses a debugger configured to create a database of debugging information, this fails to disclose the recited limitation of debugging executable code that is itself *configured to access associated data in a data repository*.

Further, it directly follows that *Wimble* does not disclose the recited limitation of determining whether the monitored executable code has accessed the associated data in the data repository. Quite the contrary, it would make no sense for the system of *Wimble* to do so. The debugger of *Wimble* would not create the debugger database and then monitor whether the very same debugger accesses the debugger database. Access by the debugger to the debugger database would be presumed. Otherwise, the debugger would not bother to create it. On this basis alone, Applicants submit that the rejection should be withdrawn and the claims be allowed.

Furthermore, the Examiner concedes that *Wimble* fails to disclose the claimed steps of:

> if so, determining whether to display the associated data on the basis of whether the associated data is restricted data; wherein determining whether to display the associated data comprises referencing predefined access restriction rules defining at least one

> rule preventing at least a portion of the associated data from being displayed to unauthorized users; and
>
> upon determining not to display the associated data on the basis of the referenced predefined access restriction rules, outputting masking characters on an output screen indicative of the associated data without revealing a value of the associated data, whereby selected data from the data repository is concealed from a user debugging the executable code.

Claim 1. Claim 9 recites a similar limitation. In other words, if a program being debugged accesses data from a data repository that is restricted data (i.e., person debugging the program is not authorized to view certain data from the database), this information may be masked using masking characters. The material cited by the Examiner refers to debugging messages that may be "selectively enabled based on developer-selected masks, based on some specified rules of filtering." *Warmink*, 2:39-40.

However, as used in *Warmink*, a mask value is used to specify which debugging messages is output during program execution. No use of masking characters is disclosed, or would be useful using this debugging technique. That is, it makes no sense whatsoever to suggest that the debugging messages, meant to inform the programmer of aspects of program operational state, could be replaced with masking characters when displayed to a user and remain useful at all.

Furthermore, *Warmink* discloses a process where the developer is presented with the original debug message. *Warmink* describes the process of replacing debug strings in the source code of a program with unique numbers (referred to as ENUM values) as follows:

> Although the text message is the largest portion of the trace statement's argument, it is also fixed in length and content for each trace statement. The present invention exploits this fact by removing or stripping out all of these text message or fixed field portions of debug trace statement arguments, and replacing them with unique numbers. Thus, source code is first obtained (step 201). This is typically the final version of the source

> code after development phase debugging, just before the code is ready for final compiling and shipping to end users.
>
> Next, the source code is run through a debug strip tool or program in accordance with the present invention, to identify each trace statement and its fixed text string portion. This debug strip program may be run on a PC (e.g., the same PC on which programming and debugging has been performed to write the program code) and provided with the file name of the completed source code. Each of the fixed text strings in the argument portions of trace statements in the source code is replaced with a unique number, such as an ENUM (step 203). This involves deleting the text string, inserting the next ENUM in the sequence (and incrementing the ENUM to the next one as necessary), and saving both the text string and the ENUM corresponding to that text string. The variable data portions of the output string are not stripped out but remain, with the ENUM, in the output string.

Warmink, 6:35-60. This passage describes a process of identifying text-string debugging messages in the source code of a program and replacing the text strings with "a unique number" The unique number does not prevent or conceal the debugging message, in fact the opposite is the case. *Warmink* goes on to describe how the "unique numbers" are used to generate a debug messages presented to a developer. When a "debug PC" is attached to a "debug port." Specifically, *Warmink* teaches:

> the "debug pc converts the debug value to a corresponding text string using a "number to string mapping table. The text string, not the ENUM value, is thus inserted in the appropriate place in the debug output string of data and displayed on monitor 126 along with other data of the debug output string.

7:65-67 – 8:1-3. In other words, the original debug string message is displayed to the user. Thus, Applicants contend that the "mask value" used in *Warmink* to filter which debug messages are displayed on a "debug PC" in no way discloses the claimed method step of "outputting masking characters on an output screen indicative of the associated data without revealing a value of the associated data, whereby selected data from the data repository is concealed from a user debugging the executable code." Plainly, nothing is concealed, and no part of the debug message is the developer restricted from viewing.

Accordingly, for all the foregoing reasons, Applicants submit that claims 1, 9, and the claims dependent therefrom, are patentable over *Wimble* in view of *Warmink*.

Regarding claim 16, the Examiner suggests that *Wimble*, in view of *Warmink*, discloses a computer-readable medium containing a debug program which, when executed, performs an operation of debugging code configured to access associated data in a repository. However, Applicants contend that *Wimble*, in view of *Warmink*, fails to disclose at least the recited limitation of "a debug engine configured to selectively pass data to the debugger user interface according to predefined access restriction rules defining at least one rule prohibiting at least a portion of the associated data from being displayed to a user operating the debug program, whereby selected data from the data repository is concealed from the user debugging the executable code."

The Examiner concedes that *Wimble* fails to disclose this limitation, but asserts that *Warmink* does. For all the reasons given above, however, Applicants submit that the passages cited from *Warmink* fail to teach or suggest a debugger interface configured to prohibit "at least a portion of the associated data from being displayed to a user operating the debug program, whereby selected data from the data repository is concealed from the user debugging the executable code." Rather, as discussed above, *Warmink* discloses a debugger user interface that allows a mask to filter which embedded debug messages (or codes representing such messages) are output during the execution of a program. Plainly, the debugging messages are not selected data from a data repository, as the debugging messages are embedded within the program code. Accordingly, Applicants assert that *Wimble*, in view of *Warmink*, fails to teach or suggest the limitations recited by claim 16.

Therefore, for all the foregoing reasons, claims 1, 9, 16, and the claims dependent therefrom are believed to be allowable, and allowance of these claims is respectfully requested.

Claims 8, 14 and 20 stand rejected under 35 U.S.C. § 103(a) as being unpatentable over *Wimble* in view of *Warmink* and further in view of *Kolawa* (U.S. Pat. 6,085,029).

Claims 8, 14, and 20 are rejected under 35 U.S.C. 103(a) as being unpatentable over *Wimble* and *Warmink* in further view of *Kolawa* (U.S. 6,085,029). Applicants respectfully traverse this rejection. Claims 8, 14, and 20 depend from one of claims 1, 9 or 16, and are therefore believed to be allowable for the reasons provided above. Accordingly, withdrawal of this rejection is respectfully requested.

Therefore, for all of the foregoing reasons, the claims are believed to be allowable, and allowance of the claims is respectfully requested.

Conclusion

Having addressed all issues set out in the office action, Applicants respectfully submit that the claims are in condition for allowance and respectfully request that the claims be allowed.

If the Examiner believes any issues remain that prevent this application from going to issue, the Examiner is strongly encouraged to contact Gero McClellan, attorney of record, at (336) 643-3065, to discuss strategies for moving prosecution forward toward allowance.

Respectfully submitted, and
S-signed pursuant to 37 CFR 1.4,

/Gero G. McClellan, Reg. No. 44,227/
Gero G. McClellan
Registration No. 44,227
PATTERSON & SHERIDAN, L.L.P.
3040 Post Oak Blvd. Suite 1500
Houston, TX 77056
Telephone: (713) 623-4844
Facsimile: (713) 623-4846
Attorney for Applicants